Pattern Analysis and Applications (2021) 24:1441–1449
https://doi.org/10.1007/s10044-021-00980-2

ORIGINAL ARTICLE

A new method of hybrid time window embedding with transformer-based traffic data classification in IoT-networked environment

Rafał Kozik[1,2] · Marek Pawlicki[1,2] · Michał Choraś[1,2]

Received: 15 January 2021 / Accepted: 29 April 2021 / Published online: 12 May 2021

Abstract
The Internet of Things (IoT) appliances often expose sensitive data, either directly or indirectly. They may, for instance, tell whether you are at home right now or what your long or short-term habits are. Therefore, it is crucial to protect such devices against adversaries and has in place an early warning system which indicates compromised devices in a quick and efficient manner. In this paper, we propose time window embedding solutions that efficiently process a massive amount of data and have a low-memory-footprint at the same time. On top of the proposed embedding vectors, we use the core anomaly detection unit. It is a classifier that is based on the transformer's encoder component followed by a feed-forward neural network. We have compared the proposed method with other classical machine-learning algorithms. Therefore, in the paper, we formally evaluate various machine-learning schemes and discuss their effectiveness in the IoT-related context. Our proposal is supported by detailed experiments that have been conducted on the recently published Aposemat IoT-23 dataset.

Keywords Deep learning · Transformers · Anomaly detection

1 Introduction

In March of 2019, only two months after a similar attack on Altran Technologies, the LockerGoga ransomware was used against Norsk Hydro, the largest aluminum manufacturer in Europe, hiring over 35000 people and having sites in more than 50 countries all across the globe. The attack caused a serious decrease in production and issues with the execution of the ongoing contracts. The losses were estimated to equal millions of dollars per day, and the grand total of losses was estimated to reach hundreds of millions of dollars. The attack occurred on 18/19 March 2019, mostly impacting the infrastructure in Norway, and other countries to a lesser extent. It resulted in the shutdown of the global Norsk Hydro network.

The attack affected work at the offices (causing, for example, problems with order documentation) as well as the industrial manufacturing, where, besides other issues, the manufacturing drivers had to be uploaded manually through usb drives.

The attack was a cyber-criminal case committed for financial gains. The ransomware had turned off a part of the system's security mechanisms, as well as the data backup processes, before starting data encryption. All the local user passwords were changed.

The 'Ransom' was not paid, the recovery of data from backups took months. As of March 2019, the LockerGoga ransomware was undetectable by 67 of the state-of-the-art antiviruses. The experts noted that better anomaly detection systems could have prevented the incident.

In June 2019, a vulnerability in the Amazon Ring Video Doorbell was discovered. The flaw in the product's security made it possible to connect to the home WiFi and possibly exploit other connected devices [2]. A similar issue was discovered in the Amazon Blink XT2 security camera. The security flaw, which was discovered in August 2019, allowed unauthorised users to view the footage from the cameras and listen to their audio. In fact, the flaw made it extremely easy to gain root access to the device [16].

✉ Rafał Kozik
rkozik@utp.edu.pl

[1] UTP University of Science and Technology, Bydgoszcz, Poland

[2] ITTI Sp. Z o.o., Poznań, Poland

The 'Attack Landscape' report illustrates that a number of network attacks are carried over Telnet and Secure Shell (SSH) with a high probability of targeting IoT devices [7].

Therefore, in this paper, we propose a new innovative method to detect anomalies in IoT environment.

The major contribution of this work is the proposition of a time window embedding solution with a transformer-based classification scheme.

The remainder of this paper is structured as follows: In Sect. 2, the related work is overviewed, in Sects. 3- 5 the proposed method is described, experimental setup is presented in Sect. 6, while the results are reported in Sect. 7. Conclusions are drawn thereafter.

2 Related work

In the literature, there are two approaches to intrusion detection, namely the signature-based and anomaly-based ones. Typically, when the attack is deterministic, one can develop a signature that will allow for its detection. However, nowadays attackers use various obfuscation techniques to evade such detection mechanisms. Therefore, the cybersecurity community is investing its efforts in the anomaly detection system. These turn to be more effective in detecting new and unknown (so called 0-day) cyber-attacks [21].

In [3], the authors performed a survey of current tendencies in cybersecurity and concluded that two major trends emerge - one is that old, proven methods are still in use in many applications. The other is that machine-learning-based (ML) approaches are increasingly more prominent. Furthermore, [17] points out that ML is now used on both the malware and security sides.

When it comes to network traffic analysis, two popular approaches are used by experts from the cybersecurity domain. One is based on deep packet inspection [4], while the other relies on network flows analysis [11]. One of the most popular protocols for network flow data collection is NetFlow [6]. That kind of data is often captured by Internet service providers for auditing and performance monitoring purposes. NetFlow samples do not contain much of sensitive data and therefore are widely available. However, the disadvantage is that such samples do not contain the raw content of network packets. Such details are valuable and can improve the effectiveness of malware detection. However, these are rarely available because of the encryption, which is often utilized by the end-points terminal.

The current research shows that the network flow data can be effectively analyzed using various machine-learning techniques such as unsupervised clustering [8], Random-Forests (RF) [22], or deep learning [19]. The authors of [5] present a range of deep neural network topologies and test the influence of hyperpaprameter setups on the accuracy of the solution. On the flip-side, in [12], a stream processing framework capable of employing a range of ML algorithms for intrusion detection is presented.

Obviously, the different methods vary in the way they process the NetFlow data. For instance, in [10], the authors proposed a solution called CCDetector. It uses a state-based behavioral model of the Command and Control channels. The author of this algorithm adapts the Markov Chain to model malware behavior and to detect similar traffic in unknown real networks. The difference from the BClus (and our approach) is the fact that instead of analyzing the complete traffic of an infected computer as a whole, the authors separate each individual connection from each IP address and treat these as an independent connection. The results obtained with this method are very promising. However, one of the concerns is the complex and time-consuming learning phase.

In opposite to that, in [19], the authors have adapted recurrent neural networks (RNN) with long short-term memory (LSTM) units on top of the NetFlow data. In addition to that they also used a flexible distributed architecture to handle the curation of large amount of data.

An interesting approach, which maps the NetFlow data to the image representation, has been presented in [13]. In order to construct the images, the authors have used such techniques as feature correlation analysis and correlation matrices. The images have been analyzed with a convolution neural network (CNN) in order to detect intrusions. According to the authors, this method achieves high accuracy.

A CNN for flow-based malware detection is also proposed in [20]. The authors advocate that current detection systems are overreliant on certain network features, like the port number, which could introduce a blind spot in the system. Thus, they calculate 35 features with the use of Netmateto to fully express the state of the network and provide those to the CNN and other ML algorithms.

The authors of [18] present a deep network model capable of automatic feature extraction, which takes time-related characteristics into consideration. To achieve that a GRU network along with a multilayer perceptron (MLP) is used. The authors also test a network with LSTM cells.

The authors of [14] evaluate autoencoders (sparse, denoising, contractive, convolutional), LSTM, and CNN for network intrusion detection. Autoencoders obtain the latent representation of the feature set. When the hidden layer has fewer neurons than the input/output layers, it is called a bottleneck, discriminative, coding, or abstraction layer. Using such a bottleneck forces the topology to acquire the most significant features.

In [9], instead of flow classification, the flow prediction approach is used. In order to achieve this, the authors combine an RNN (with gated recurrent units) with the so-called linear regression layer, which allows for producing prediction in a similar fashion as auto-regressive integrated moving average (ARIMA) models do with time series.

In [1], the authors used the auto-regressive fractionally integrated moving average (ARFIMA) model and proposed the Hyndman-Khandakar algorithm to estimate the polymonials parameters and the Haslett and Raftery algorithm to estimate the differencing parameters for network anomaly detection.

3 Proposed method

The proposed solution (see Fig. 1) captures network flows (as streams), calculates feature vectors over a predefined time window, provides these vectors to a binary classifier, which eventually produces the detection output (benign for normal traffic or anomaly for traffic containing suspicious patterns). In the next section, the details on each of these processing steps will be provided. First, the overview of the input data is given, and then, the effective methods for feature extraction are elaborated upon. Finally, a brief description of the classification methods incorporated in this work is provided.

3.1 Flow-based data acquisition

Conceptually, in this approach, the data are collected from the network in the form of communication flows traversing such devices as switches, routers, or hosts. This kind of data captures aggregated network properties and statistics. From the architectural point of view, the network traffic going through the flow-enabled devices is collected and later on sent to the collectors - the network elements, which store them and keep them for the operator for later analysis. In particular, network flows are often used by the network administrators for auditing purposes. A single flow aggregates such characteristics as:

- incoming and outgoing number of bytes
- IP addresses taking part in the communication
- utilized source and destination ports
- utilized type of protocol (e.g., Transmission Control Protocol (TCP) or User Datagram Protocol (UDP))

(e.g., the number of bytes sent and received) about packets that have been sent by a specific source address to a specific destination address. It is obvious that it must be possible to identify some patterns of anomalous behavior of network nodes from such kind of data. Some of these patterns may be related to malware infection or help the network administrator to identify adversaries.

3.2 Time window embedding with probabilistic data structures

The rationale behind the proposed embedding is to encode a network flow using only its nearest neighborhood in the time domain. This approach allows us to capture some short-term malicious behavior of specific network elements and nodes.

In the proposed approach, we calculate the statistical properties of a group of flows that have been collected for a specific source IP address within short and fix-length time spans called time windows.

As it was presented in the previous section, a single flow exhibits various characteristics describing the two-way communication (e.g., number of flows, destination IP, etc.). For each of these characteristics, it is possible to calculate various statistical properties such as mean, median value, min/max values. In order to make it clear what is being calculated for each of those characteristics, we have included Table 1. Overall, it leads to the situation where the traffic characteristics produced by a single IP address within the considered tumbling time window are described by 37 values.

In general, counting the number of flows and/or accumulating the sum of inbound and outbound packets may be trivial, but the situation is different when it comes to distinct counting or finding the most frequent element (e.g., destination port) in the stream of data. The straightforward approach would be to maintain a dynamic list. Whenever a new element is retrieved from the stream, one must scan the entire list and check whenever that element is there. If not, the list needs to be resized and new element added. Moreover, there is another level of complexity when we want to

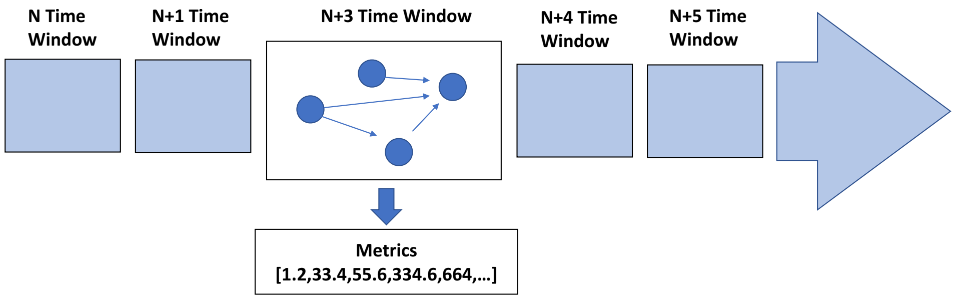

Fig. 1 Tumbling windows - example of window statistics embedding

Table 1 Overview of all the features extracted from network flows

	Min	Max	Avg	Median	Unique count	Most frequent
Number of flows	+	+	+	+		
Destination port					+	+
Destination address					+	+
Protocol					+	+
Inbound packets	+	+	+	+		
Outbound packets	+	+	+	+		
Total packets	+	+	+	+		
Inbound bytes	+	+	+	+		
Outbound bytes	+	+	+	+		
Total bytes	+	+	+	+		
Port-protocol pair						+
IP-port pair						+
IP-port-prot. triplet						+

merge the results obtained from two concurrent processes that perform distinct counting.

There is a class of data structures which are known as probabilistic (or sketchy). These have the ability to describe remarkably large sets with sub-logarithmic or constant space complexity. That implies that there is no need to scale-up the data processing system when it undergoes the transition from thousands, to millions, or even billions of records that need to be analyzed.

Probabilistic data structures rely on various mechanisms to compress data, and often, these mechanisms may cause them to contain inaccurate information.

However, this inaccuracy should not have a strong impact on the detection part. The assumption is based on the observation that the classifiers can handle such changes to some extent and be able to return the correct decision. It must be noted that here we are talking about changes that are within a range of 1–2% for one of the features building the vector.

Probabilistic data structures have several advantages. First of all, the size of such structures grows significantly slower with respect to the input data. In many cases, it is orders of magnitude smaller. Moreover, it is also possible to make the trade-off between the accuracy of prediction vs. the size of the data structure. They are naturally suited for measuring network traffic, which has a form of streaming data, where each item in the stream needs to be analyzed quickly and in needs to update a data structure that summarises some properties (e.g., the number of distinct IP addresses or the most frequently used service). A substantially useful property of the probabilistic data structures is the ability to be merged. It means that when the stream is split into two parts and the summary over them is calculated separately, the result will be the same as if it was calculated over the entire (original) stream. As a result, this makes the probabilistic structure highly parallelizable and suitable for distributed computing platforms (e.g., Hadoop, Spark, Druid, etc.)

4 Frequent items and distinct counting - the problem overview

In order to calculate the most frequent destination port or destination host (or even concatenation of both) originating from a specific IP address, one may use data structures such as a hash table to accomplish that. In such a case, the new item is put in the hash table and the counter for that item is set to 0. Whenever the entry in the hash table already exists, a counter is just incremented. However, such an approach may quickly become impractical when the amount of input data is significant, because of two reasons. Firstly, along with the growing size of the input data, the hash table will grow as well and eventually its capacity will exceed the amount of available RAM memory. Secondly, the collisions in the hash table are handled as a linked list. It means that whenever a new item is hashed to the bucket that is already taken, it will be appended (linked) after the existing one. Therefore, when the list becomes longer and longer, the access time to such elements in the hash table will be substantially longer as well. Also, the dynamic allocation of the memory (for a new element) is also time consuming. In this section, the most important sketchy data structures that have been used in this research are described.

4.1 Count-min data structure

Count-min (CM) data structure allows for counting of items that are of a different type, e.g., how many times a specific IP address has contacted port 8080. From the scientific point of view, CM is an array of width w and depth d $CM[1, 1] \dots CM[d, w]$. It uses a set of d hash functions:

$$h_1 \dots h_d : \{1 \dots n\} \rightarrow \{1 \dots d\} \tag{1}$$

which belong to a random pairwise-independent family of functions. The width w and depth d are chosen based on allowed by user error rates and are calculated as:

$$w = \left\lceil \frac{e}{\epsilon} \right\rceil \quad d = \left\lceil ln\frac{1}{\sigma} \right\rceil \tag{2}$$

where ϵ (an acceptable error in estimation) is the error factor and σ is the error probability. At the beginning, the CM array is initialized with 0 values. Each time the count needs to be updated for a specific value x, the hash functions are calculated and modded with the width w. This yields the column number $col = h_n(x)\%w$. Finally, the cell at position (n, col) in the CM array is incremented by one. We use a similar approach when querying the data structure. We simply take the modded values obtained from hash functions and find the minimum. The visual representation of the concept behind the count-min data sketch is presented in Fig. 2.

4.2 HyperLogLog sketch

HyperLogLog (HLL) belongs to a family of algorithms that aim at estimating the cardinality of a dataset. It relies on the probabilistic counting method. Assuming that we have a large data set with duplicated entries, we can evenly distribute the elements in the dataset using a hashing function and estimate the cardinality using the hashed values. The common approach is to count the leading zeros in the binary representation of the hashed values. The probability of observing n leading zeros is equal to $\frac{1}{2^n}$. In other words, if we denote $p(v_1)$ as a number of leading zeros in v_1, we can calculate the cardinality as $n = 2^R$, where:

$$R = max(p(v_1), p(v_1), \dots, p(v_m)) \tag{3}$$

The visual representation of the HyperLogLog data sketch is presented in Fig. 3.

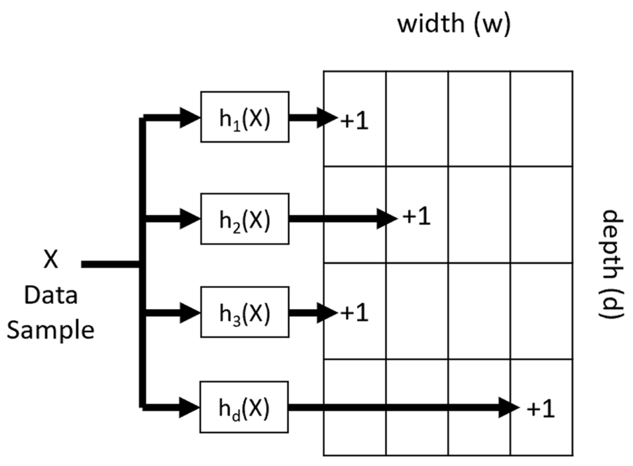

Fig. 2 Count-min data structure - overview of the architecture

Obviously, a single estimator of that kind is subject to high variance. Therefore, the common approach is to use several estimators and to average the results. This can be achieved using several independent hash functions.

5 Classification with transformer

Transformers have been proposed in the area of natural language processing (NLP). This is a relatively novel architecture that aims at solving sequence-to-sequence problems while handling long-range dependencies. The original transformer architecture involves the so-called encoding and decoding parts. In this research, only the encoder is used (see Fig.4), since the aim is to encode the behavior of specific node elements using the latent representation produced by the encoder part of the transformer architecture.

The data are ingested into the transformer using the time windows embedding technique described in the previous section. In general, several network flows belonging to a specific time window and related with a specific IP address are encoded using the technique leveraging probabilistic data sketches. This operation results in vectors of a fixed length. The transformer works with sequences, which in our case describe the behavior of a specific network element over the defined time period. The sequence is composed by putting several embedding vectors one next to another.

Next, the sequence of vectors goes through the positional encoding, which allows us to capture the order of the vector in the input sequence. This looks quite useful from our perspective, because usually the attacker executes some series of actions in order to carry out a successful cyberattack.

Afterward, the positionally encoded input reaches the multiheaded attention layer. From a high perspective, this layer allows the model to look at other positions in the input sequence for clues that can improve the final detection. For example, the infected machine would try to contact the botmaster if a few moments before it was infected with malware.

In the next step, the information coming from the self-attention is normalized and goes to the feed-forward layer, the output of which is normalized as well. As it is depicted in the diagram, there are two residual connections: one around the self-attention and the other around the feed-forward layer.

Because the input leaving the transformer layer contains one output vector for each element in the input sequence, we use the average pooling layer. It takes the mean across all the elements in the input. Finally, on top of the entire model, we used two layers of the feed-forward network with two outputs, one to indicate the benign sequence and the other to indicate the anomalous sequence.

6 Experiments

6.1 The goal of the experiments

The goal of the experiments is to compare the proposed approach with the state-of-the-art methods. In this paper, we have considered two different evaluation scenarios. Firstly, we investigate how the proposed time window embedding technique operates along with the transformer-based anomaly detection architecture. In that regard, we have compared our approach with RandomForest (with 500,100,10 trees, respectively), vanilla REPTree (decision/regression tree), and AdaBoosted version of REPTree. Secondly, we investigate to what extends the proposed model is able to generalize to unknown malware infection scenarios. In that regard, we have considered various test-case scenarios where the models are trained and evaluated on malicious samples that have been recorded for different attacks (or infections).

6.2 Aposemat IoT-23 dataset

IoT-23 is a dataset of network traffic from Internet of Things (IoT) devices. It has 20 malware captures executed in IoT devices, and three captures for benign IoT devices traffic. It was first published in January 2020, with captures ranging from 2018 to 2019. This IoT network traffic was captured in the Stratosphere Laboratory, AIC group, FEL, CTU University, Czech Republic. Its goal is to offer a large dataset of real and labelled IoT malware infections and IoT benign traffic for researchers to develop machine-learning algorithms. This dataset and its research is funded by Avast Software, Prague. In addition to easier reproducibility of the study, using a benchmark dataset allows to handle various privacy issues, as outlined in [15].

6.3 Experimental protocol

The dataset was split into training and testing parts. This is explained in Table 2. Different scenarios are used, where different parts of the original dataset have been employed. We followed such an approach in order to prove that the proposed method can generalize well to unknown malware families. In that regard, we have used different scenarios (malware families) for training and testing the models. Nonetheless, some malware names appear both in training and testing. However, these malware are recorded in different network captures, which concern different contexts (different network elements, different IoT devices, etc.)

Additionally, for the training part of the dataset, five-fold cross-validation was adapted. This allowed to calculate the standard deviation of measured performance characteristics. Moreover, this also enables the discussion on the significance of differences obtained for various approaches. For that matter, *t-test* statistical hypothesis test was used.

Table 2 IoT23 dataset - scenarios setup (x and – indicate training and validation sets, respectively)

	Scenario				Malware
	1	2	3	4	
CTU-Honeypot-Capture-4-1	x	x	x	x	Benign
CTU-Honeypot-Capture-5-1	x	x	x	x	Benign
CTU-Honeypot-Capture-7-1	x	x	x	x	Benign
CTU-IoT-Malware-Capture-34-1	x	x	x	x	Mirai
CTU-IoT-Malware-Capture-43-1	x	x	x	x	Mirai
CTU-IoT-Malware-Capture-1-1	x	x	x	x	Hide&Seek
CTU-IoT-Malware-Capture-3-1	x	x	x	x	Muhstik
CTU-IoT-Malware-Capture-35-1	x	x	x	x	Mirai
CTU-IoT-Malware-Capture-39-1	x	x	x	-	IRCBot
CTU-IoT-Malware-Capture-7-1	x	x	x	-	Mirai
CTU-IoT-Malware-Capture-8-1	x	x	x	-	Hakai
CTU-IoT-Malware-Capture-9-1	x	x	x	-	Hajime
CTU-IoT-Malware-Capture-20-1	x	x	x	-	Torii
CTU-IoT-Malware-Capture-21-1	x	x	x	-	Torii
CTU-IoT-Malware-Capture-42-1	x	x	x	-	Trojan
CTU-IoT-Malware-Capture-17-1	x	x	-	-	Kenjiro
CTU-IoT-Malware-Capture-36-1	x	x	-	-	Okiru
CTU-IoT-Malware-Capture-33-1	x	-	-	-	Kenjiro
CTU-IoT-Malware-Capture-48-1	x	-	-	-	Mirai
CTU-IoT-Malware-Capture-44-1	-	-	-	-	Mirai
CTU-IoT-Malware-Capture-49-1	-	-	-	-	Mirai
CTU-IoT-Malware-Capture-52-1	-	-	-	-	Mirai
CTU-IoT-Malware-Capture-60-1	-	-	-	-	Gagfyt

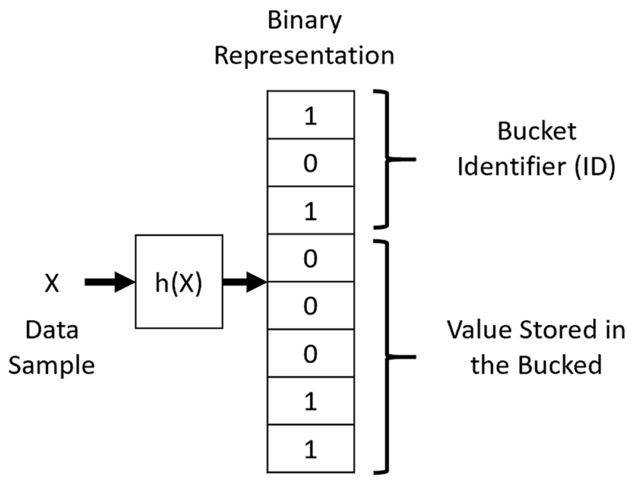

Fig. 3 Architecture of HyperLogLog data structure

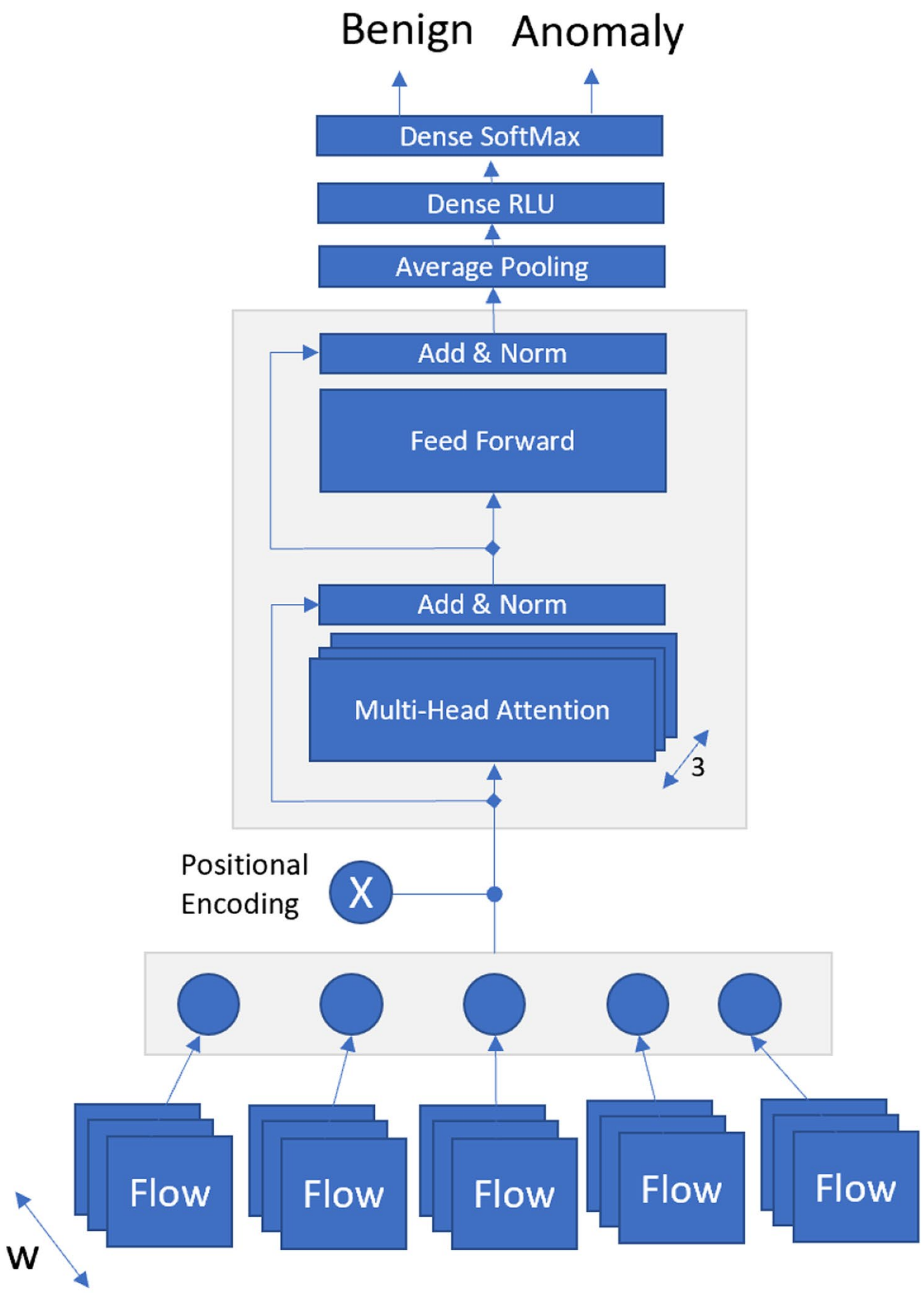

Fig. 4 The architecture of the proposed transformer-based classifier

6.4 Evaluation metrics

Before using various machine-learning algorithms, the raw network flows are processed in order to produce the time window embedding vectors. The procedure is detailed in the previous sections. The procedure for calculating metrics is as follows:

1. communication flows are aggregated into time windows (here we have used 3 min. time windows).
2. for given time, windows embedding vectors are calculated
3. within the ground-truth communication flows, labels are examined against the predicted ones and the TP, TN, FP, and FN errors (true and false positives and negatives) are measured.
4. finally, Recall, Precision, F-measure Rate are estimated and reported.

7 Results

The results for classification parts are presented in Table 3. We have compared the proposed transformer-based approach against various and popular machine-learning techniques. On the list, there is a decision tree and classifiers ensembles (one is AdaBoost, while the other is the well-known RandomForest).

The *t-test* statistical hypothesis test was used to validate that the results obtained by the proposed approach are significantly different from the other compared methods. In that regard, the values followed *by* ± *symbol* (Table 3) indicate standard deviation.

It must be noted that we have used different scenarios for training and testing the models. This proves the approach can generalize well to unknown malware families.

What can be found in Table 3 is that these base-line methods behave quite well. However, the transformer-based approach outperforms other methods in most of the considered scenarios. It is quite vivid for *f*1-*score* metric that has been visually presented in Fig.5. It must be noted that the transformer-based approach allowed us to achieve remarkably good results for the fourth scenario, where less than 40% of the original datasets is used.

The second good results are reported for the RandomForest. The experiments show that increasing the number of trees above 100 does not bring much to the effectiveness. The Adaptive Boosting (AdaBoost) ensemble of REPTrees performs well for the first and the second scenarios; for the scenarios 3 and 4, this method stays a bit behind the RandomForest.

What can be found interesting is the observation that the f1-score does not change that much between scenarios 1,2,3 and 4 for the proposed method. On the other hands, one can observe quite significant fluctuations for the other approaches. In particular, this change is much more significant for Reduced Error Pruning Decision Tree (REPTree) and Adaptive Reduced Error Pruning Decision Tree (AdaREPT) classifiers.

Table 3 Effectiveness comparison for different classifiers and scenarios

Scenario	Method	Precision	Recall	f1-score
1	Proposed method	0.902 ± 0.001	0.967 ± 0.005	0.934 ± 0.003
2		0.912 ± 0.003	0.940 ± 0.008	0.926 ± 0.005
3		0.916 ± 0.003	0.977 ± 0.002	0.945 ± 0.002
4		0.910 ± 0.002	0.964 ± 0.004	0.936 ± 0.002
1	RF500	0.866 ± 0.001	0.928 ± 0.001	0.896 ± 0.001
2		0.858 ± 0.001	0.803 ± 0.000	0.829 ± 0.001
3		0.866 ± 0.000	0.877 ± 0.000	0.872 ± 0.000
4		0.859 ± 0.001	0.930 ± 0.000	0.893 ± 0.001
1	RF100	0.877 ± 0.002	0.943 ± 0.001	0.909 ± 0.001
2		0.861 ± 0.002	0.791 ± 0.004	0.825 ± 0.001
3		0.871 ± 0.003	0.873 ± 0.000	0.872 ± 0.002
4		0.867 ± 0.001	0.928 ± 0.003	0.896 ± 0.002
1	RF10	0.838 ± 0.002	0.814 ± 0.003	0.826 ± 0.002
2		0.803 ± 0.003	0.670 ± 0.005	0.730 ± 0.003
3		0.861 ± 0.003	0.909 ± 0.001	0.884 ± 0.001
4		0.846 ± 0.007	0.826 ± 0.002	0.836 ± 0.004
1	REPTree	0.833 ± 0.004	0.762 ± 0.001	0.796 ± 0.002
2		0.846 ± 0.005	0.717 ± 0.002	0.776 ± 0.002
3		0.808 ± 0.005	0.766 ± 0.003	0.786 ± 0.003
4		0.844 ± 0.003	0.945 ± 0.003	0.892 ± 0.002
1	AdaBoost	0.813 ± 0.002	0.763 ± 0.003	0.788 ± 0.002
2		0.836 ± 0.006	0.643 ± 0.003	0.727 ± 0.004
3		0.819 ± 0.005	0.710 ± 0.003	0.761 ± 0.003
4		0.807 ± 0.003	0.697 ± 0.003	0.748 ± 0.001

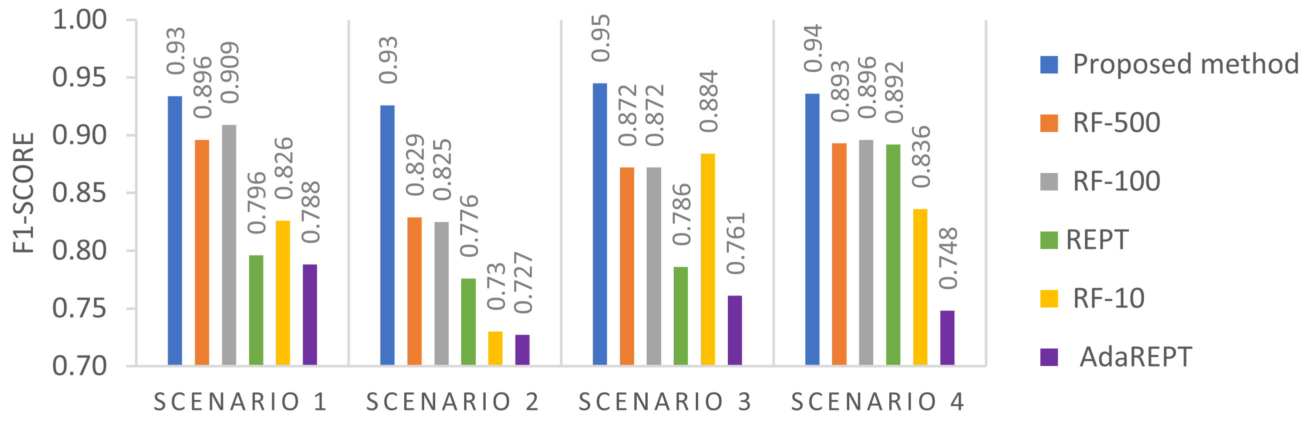

Fig. 5 Comparison of the f1-score achieved for various algorithms and scenarios

8 Conclusions

In this paper, we propose innovative anomaly detection that utilizes innovative time windows embedding solutions that efficiently process a massive amount of data, while having a low-memory-footprint at the same time. The core anomaly detection is based on the transformer's encoder unit followed by a two-layer feed-forward neural network. In the paper, we have formally evaluated various machine-learning schemes in order to compare these with the proposed approach and to discuss their effectiveness in the IoT-related context. The proposal is supported by detailed experiments that have been conducted on the recently published Aposemat IoT-23 dataset. Our experiments show that the proposed approach that leverages the transformer-based classification performs best in most of the considered scenarios.

Acknowledgements This work is funded under PREVISION project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 833115.

References

1. Andrysiak T, Saganowski Ł, Choraś M, Kozik R (2014) Network traffic prediction and anomaly detection based on arfima model. In: International Joint Conference SOCO'14-CISIS'14-ICEUTE'14, pp. 545–554. Springer
2. BitDefender: Ring video doorbell pro under the scope (2019). https://www.bitdefender.com/files/News/CaseStudies/study/294/Bitdefender-WhitePaper-RDoor-CREA3949-en-EN-GenericUse.pdf
3. Caviglione L, Choraś M, Corona I, Janicki A, Mazurczyk W, Pawlicki M, Wasielewska K (2020) Tight arms race: overview of current malware threats and trends in their detection. IEEE Access
4. Cheng Z, Beshley M, Beshley H, Kochan O, Urikova O (2020) Development of deep packet inspection system for network traffic analysis and intrusion detection. In: 2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), pp. 877–881
5. Choraś M, Pawlicki M (2020) Intrusion detection approach based on optimised artificial neural network. Neurocomputing
6. Claise B (2004) Cisco systems netflow services export version 9. rfc 3954 (informational)
7. F-Secure: the f-secure attack landscape report H1-2020 (2020). https://www.f-secure.com/content/dam/press/de/media-library/reports/F-Secure-attack-landscape-h12020.pdf
8. Flanagan K, Fallon E, Awad A, Connolly P (2017) Self-configuring netflow anomaly detection using cluster density analysis. In: 2017 19th International Conference on Advanced Communication Technology (ICACT), pp. 421–427
9. Fu R, Zhang Z, Li L (2016) Using lstm and gru neural network methods for traffic flow prediction. In: 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), pp. 324–328
10. Garcia S (2014) dentifying, modeling and detecting botnet behaviors in the network. Ph.D. thesis, Instituto Superior de Ingenier'ıa de Software Tandil Departamento de Computacio'n y Sistemas
11. Hardegen C, Pfülb B, Rieger S, Gepperth A (2020) Predicting network flow characteristics using deep learning and real-world network traffic. IEEE Transactions on Network and Service Management pp. 1–1
12. Komisarek M, Choraś M, Kozik R, Pawlicki M (2020) Real-time stream processing tool for detecting suspicious network patterns using machine learning. In: Proceedings of the 15th International Conference on Availability, Reliability and Security, pp. 1–7
13. Liu X, Tang Z, Yang B (2019) Predicting network attacks with cnn by constructing images from netflow data. In: 2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), pp. 61–66
14. Naseer S, Saleem Y, Khalid S, Bashir MK, Han J, Iqbal MM, Han K (2018) Enhanced network anomaly detection based on deep neural networks. IEEE Access 6:48231–48246. https://doi.org/10.1109/ACCESS.2018.2863036
15. Pawlicka A, Jaroszewska-Choras D, Choras M, Pawlicki M (2020) Guidelines for stego/malware detection tools: achieving gdpr compliance. IEEE Technol Soc Mag 39(4):60–70
16. Tenable: Blink XT2 sync module multiple vulnerabilities (2019). https://www.tenable.com/security/research/tra-2019-51
17. Thanh CT, Zelinka I (2019) A survey on artificial intelligence in malware as next-generation threats. Mendel 25:27–34
18. Xu C, Shen J, Du X, Zhang F (2018) An intrusion detection system using a deep neural network with gated recurrent units. IEEE Access 6:48697–48707. https://doi.org/10.1109/ACCESS.2018.2867564
19. Yang C, Liu J, Kristiani E, Liu M, You I, Pau G (2020) Netflow monitoring and cyberattack detection using deep learning with ceph. IEEE Access 8:7842–7850
20. Yeo M, Koo Y, Yoon Y, Hwang T, Ryu J, Song J, Park C (2018) Flow-based malware detection using convolutional neural network. In: 2018 International Conference on Information Networking (ICOIN), pp. 910–913. https://doi.org/10.1109/ICOIN.2018.8343255
21. Zaman M, Lung C (2018) Evaluation of machine learning techniques for network intrusion detection. In: NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, pp. 1–5
22. Zhang H, Dai S, Li Y, Zhang W (2018) Real-time distributed-random-forest-based network intrusion detection system using apache spark. In: 2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC), pp. 1–7